Chapter 1.20
Handling of Software Quality Defects in Agile Software Development

Jörg Rech
Fraunhofer Institute for Experiemental Software Engineering (IESE), Germany

ABSTRACT

Software quality assurance is concerned with the efficient and effective development of large, reliable, and high-quality software systems. In agile software development and maintenance, refactoring is an important phase for the continuous improvement of a software system by removing quality defects like code smells. As time is a crucial factor in agile development, not all quality defects can be removed in one refactoring phase (especially in one iteration). Documentation of quality defects that are found during automated or manual discovery activities (e.g., pair programming) is necessary to avoid wasting time by rediscovering them in later phases. Unfortunately, the documentation and handling of existing quality defects and refactoring activities is a common problem in software maintenance. To recall the rationales why changes were carried out, information has to be extracted from either proprietary documentations or software versioning systems. In this chapter, we describe a process for the recurring and sustainable discovery, handling, and treatment of quality defects in software systems. An annotation language is presented that is used to store information about quality defects found in source code and that represents the defect and treatment history of a part of a software system. The process and annotation language can not only be used to support quality defect discovery processes, but is also applicable in testing and inspection processes.

INTRODUCTION

The success of software organizations—especially those that apply agile methods—depends on their ability to facilitate continuous improvement of their products in order to reduce cost, effort, and time-to-market, but also to restrain the ever increasing complexity and size of software systems. Nowadays, industrial software development

is a highly dynamic and complex activity, which is not only determined by the choice of the right technologies and methodologies, but also by the knowledge and skills of the people involved. This increases the need for software organizations to develop or rework existing systems with high quality within short periods of time using automated techniques to support developers, testers, and maintainers during their work.

Agile software development methods were invented to minimize the risk of developing low-quality software systems with rigid process-based methods. They impose as little overhead as possible in order to develop software as fast as possible and with continuous feedback from the customers. These methods (and especially extreme programming (XP)) are based upon several core practices, such as *simple design*, meaning that systems should be built as simply as possible and complexity should be removed, if at all possible.

In agile software development, organizations use quality assurance activities like refactoring to tackle defects that reduce software quality. *Refactoring* is necessary to remove *quality defects* (i.e., bad smells in code, architecture smells, anti-patterns, design flaws, negative design characteristics, software anomalies, etc.), which are introduced by quick and often unsystematic development. As time is a crucial factor in agile development, not all quality defects can be removed in one refactoring phase (especially in one iteration). But the effort for the manual discovery, handling, and treatment of these quality defects results in either incomplete or costly refactoring phases.

A common problem in software maintenance is the lack of documentation to store this knowledge required for carrying out the maintenance tasks. While software systems evolve over time, their transformation is either recorded explicitly in a documentation or implicitly through a versioning system. Typically, problems encountered or decisions made during the development phases get lost and have to be rediscovered in later maintenance phases. Both expected and unexpected CAPP (corrective, adaptive, preventive, or perfective) activities use and produce important information, which is not systematically recorded during the evolution of a system. As a result, maintenance becomes unnecessarily hard and the only countermeasures are, for example, to document every problem, incident, or decision in a documentation system like bugzilla (Serrano & Ciordia, 2005). The direct documentation of quality defects that are found during automated or manual discovery activities (e.g., code analyses, pair programming, or inspections) is necessary to avoid wasting time by rediscovering them in later phases.

In order to support software maintainers in their work, we need a central and persistent point (i.e., across the product's life cycle) where necessary information is stored. To address this issue, we introduce our annotation language, which can be used to record information about quality characteristics and defects found in source code, and which represents the defect and treatment history of a part of a software system. The annotation language can not only be used to support quality defect discovery processes, but is also applicable for testing and inspection processes. Furthermore, the annotation language can be exploited for tool support, with the tool keeping track and guiding the developer through the maintenance procedure.

Our research is concerned with the development of techniques for the discovery of quality defects as well as a quality-driven and experience-based method for the refactoring of large-scale software systems. The instruments developed consist of a technology and methodology to support decisions of both managers and engineers. This support includes information about where, when, and in what configuration quality defects should be engaged to reach a specific configuration of quality goals (e.g., improve maintainability or reusability). Information from the diagnosis of quality defects supports maintainers in select-

PRISM: Visualizing Personalized Real-Time Incident on Security Map
Takuhiro Kagawa, Sachio Saikiand Masahide Nakamura (2018). *International Journal of Software Innovation (pp. 46-58).*
www.irma-international.org/article/prism/210454